WEST

Generate Collection Print

L6: Entry 1 of 6 File: PGPB Oct 17, 2002

DOCUMENT-IDENTIFIER: US 20020152260 A1
TITLE: Dynamic agent with embedded web server and mark-up language support for e-commerce automation

Abstract Paragraph (1):
An agent computer program for use in an automated electronic commerce infrastructure. A client-agent communication mechanism for enabling communication between an agent computer program and at least one client computer process is provided. The client-agent communication mechanism includes a web server embedded in the agent computer program that utilizes a predetermined Internet communication protocol for communication between the agent computer program and the client computer process. An inter-agent communication mechanism is provided for enabling the agent computer program to communicate with other agents. The inter-agent communication mechanism employs documents written in a predetermined markup language for communication.

Summary of Invention Paragraph (2):
[0001] The present invention is generally related to dynamic agent computer programs executable on computer systems and the like, and in particular, a dynamic agent computer programs having an embedded web server and mark-up language support for e-commerce automation.

Summary of Invention Paragraph (13):
[0010] The present invention provides an agent computer program with communication mechanisms for use in an automated electronic commerce infrastructure. A client-agent communication mechanism for enabling communication between an agent computer program and at least one client computer process is provided. The client-agent communication mechanism includes a web server embedded in the agent that utilizes a predetermined Internet communication protocol for communication between the agent computer program and the client computer process. An inter-agent communication mechanism is provided for enabling the agent computer program to communicate with other agents. The inter-agent communication mechanism employs documents written in a predetermined markup language for communication.

Detail Description Paragraph (12):
[0029] The client-agent communication mechanism 210 includes a web server 214 for providing the web pages associated to the agent to a client computer process upon request. A web server can be any server process running at a web site that sends out web pages in response to HTTP requests from remote browsers. For example, the web server 214 can be an HTTP server that processes incoming and outgoing data written according to a predetermined Internet communication protocol, such as the HyperText Transport Protocol (HTTP). Preferably, the web server 214 is embedded in the agent. By embedding the web server 214 in the agent and having one or more web pages associated with the agent, the agent program becomes a web object that can be easily accessed by other computer processes without the need of a custom interface. It is further noted that computer processes can interact with the agent computer program without having to maintain a persistent connection with the agent computer program, thereby allowing asynchronous interaction and saving system resources.

Detail Description Paragraph (21):
[0038] FIG. 6 is a flow chart illustrating communication between an agent computer program and a client computer process performed by the client-agent communication mechanism of FIG. 2 in accordance with one embodiment of the present invention. In step 610, a request (e.g., an HTTP request) is received by the client-agent communication mechanism 210. The request specifies at least one web page that corresponds to the agent by using an address, such as a URL address. In step 620, the dynamic agent then employs a web server (e.g., an embedded HTTP server) to communicate or publish the

requested web page to a browser of the client computer process. For example, the current status of the agent can be published to the client computer process via one or more web pages.

Detail Description Paragraph (23):
[0040] In step 650, the agent can optionally initiate contact with the client computer process by specifying the client's web address. One application where this may occur is approval for a proposed action (e.g., a gray area is reached in the price of the product, and the agent requests client's approval of proposed price).

Detail Description Paragraph (25):
[0042] The client-agent communication mechanism 210 allows a client to monitor, manage, or interact with the agent remotely. When a dynamic agent is configured with an embedded web server, the agent can deliver a web page to a browser on a remote computer. The web page can then be used by the client to interact with the agent remotely. One advantage of the mechanism 210 is the provision of an easy-to-use, intuitive, and flexible graphical user interface (GUI), which is especially important when the client is a human.

Detail Description Paragraph (26):
[0043] The main difference between treating agents as Web objects and as other Java objects is that each agent has a Web page, which is accessed via a URL. This Web page contains information about the agent itself and about the task it is doing. With this mechanism we can build a virtual market where clients and servers are all connected to the Web, have Web-facing representations, and are able to offer and participate in services on the Web.

Detail Description Paragraph (27):
[0044] As a Web object, an agent connected to the Internet is accessible via HTTP or XML. This typically requires that the agent embeds, or accesses, a web server. An agent is provided with at least one web page (HTML or XML page) that provides an interface for it to be accessed via any browser. The agent "publishes" on that page, a set of control, maintenance and application operations that can be accessed or invoked by using a Web browser.

Detail Description Paragraph (31):
[0048] It is noted that inter-agent communication using messages does not require the involvement of a web server. For example, the inter-agent communication facility can be an electronic mail facility that supports messages in the form of electronic mail. In this manner, electronic mail attachments, such as text, voice, video data, may also be communicated.

Detail Description Paragraph (35):
[0052] A message can include the following: 1) envelope information (e.g., name of sender, name of recipient, etc., 2) required interpreter or ontology; 3) and the contents of the message (e.g., data etc.).

Detail Description Paragraph (37):
[0054] Autonomous agents cooperate by sending messages and using concepts from domain ontology. The inter-agent communication mechanism 220 of the present invention provides a message format with meaningful structure and semantics. The ontology switching module 224 provides a mechanism for agents to exchange ontologies and message interpreters.

Detail Description Paragraph (41):
[0058] Although XML is well structured for encoding semantically meaningful information, it must be based on the ontology. As ontology varies from domain to domain, and may even be dynamic for dynamically formed domains. A significant issue is how to exchange the semantics of domain models, and how to interpret messages differently in different problem domains. The ontology switching module 224 of the present invention is provided to address this issue and is discussed in greater detail hereinafter.

Detail Description Paragraph (42):
[0059] Generally, a domain ontology provides a set of concepts, or meta-data, that can be queried, advertised and used to control the behavior of cooperating agents. These concepts can be marked using XML tags, and then a set of commonly agreed tags, underlie message interpretation. The structures and the semantics of the documents used in a particular problem domain are represented by the corresponding DTDs and interpreters.

Detail Description Paragraph (48):
[0065] Ontology Model Switching

Detail Description Paragraph (49):
[0066] Different application domains have different ontology models with different agent communication languages and language interpreters although they are in XML format. In a particular application domain, agents communicate using domain specific XML language constructs and interpreters.

Detail Description Paragraph (50):
[0067] The inter-agent communication mechanism 220 of the present invention enables a dynamic agent to participate in multiple applications. For example, the dynamic agent communicates with other agents for the business of one domain (D.sub.a) by employing D.sub.a's language and language interpreter. Similarly, for the business of another domain (D.sub.b) by employing D.sub.b's language and language interpreter. A dynamic agent can carry multiple interpreters and adapts to different application domains and ontologies by employing the ontology switching module 224 to detect the particular domain and automatically switch the DTD's and interpreters, thereby enabling communication with agents in that domain.

CLAIMS:

1. A method of communicating between a client computer process and an agent computer program having an embedded web server comprising the steps of: a) receiving a request for at least one web page associated with the agent computer program; and b) in response to the request, using the web server to provide the requested web page for use by the client computer process to receive information from the agent computer program or to issue an instruction to the agent computer program.

2. The method of claim 1 further comprising: using the web server to provide the requested web page for use by the client computer process to issue an instruction to the agent computer program; and modifying the behavior or status of the agent computer program based on the instruction.

14. The agent computer program of claim 13 wherein the mechanism for enabling communication between the agent computer program and at least one other computer process includes a web server embedded in the agent computer program for using a predetermined Internet communication protocol to communicate with the computer process; wherein the web server processes incoming and outgoing data that is formatted according to the predetermined Internet communication protocol; and at least one web page associated with the agent computer program for use by a computer process to communicate information therewith.

WEST

End of Result Set

Generate Collection | Print

L16: Entry 2 of 2 File: PGPB Sep 19, 2002

DOCUMENT-IDENTIFIER: US 20020133504 A1
TITLE: Integrating heterogeneous data and tools

Abstract Paragraph (1):
A distributed data processing system may include an interface that receives a data processing request from a requesting entity, a processing server to provide access to local data processing applications, a shadow processing server to provide access to remote data processing applications, and an application server to fulfill the received data processing request by selectively accessing local and remote data processing applications transparently to the requesting entity. Access to data may be facilitated by providing heterogeneous data sources with software wrappers that provide an object representation of the data source, providing outputs of software wrappers to a first accumulator that aggregates data to generate a first aggregate data representation, and using a second accumulator to generate a second aggregate data representation based on the first aggregate data representation from the first accumulator. The software wrappers may hide details (e.g., format, location) of the data source.

Summary of Invention Paragraph (16):
[0015] U.S. Pat. No. 6,125,383 discloses a research system that employs Java.TM. and Common Object Request Broker Architecture (CORBA) technology in order to integrate biological and/or chemical data with individual analysis tools resident on a local server.

Summary of Invention Paragraph (20):
[0019] IBM's Garlic technology is a middleware system that employs data wrappers to encapsulate data sources. These data wrappers mediate between the middleware and the data sources. After receiving a search request, the query execution engine works with the wrappers to determine the best search scheme across all the data sources for the data sources as a whole, not each individual data source. The wrapper may execute the query using Structured Query Language (SQL) statements. The Garlic technology is incorporated into IBM's biosciences software package Discovery Link.

Summary of Invention Paragraph (22):
[0020] The systems and techniques described here may provide tools useful for the integration and analysis of data from disparate, heterogeneous sources and formats. One implementation includes a platform in which integrated data is normalized, duplicate data entries are erased, and consistent tenninology is used to describe the data. The platform can be written entirely in a Java programming language and environment and may be compatible with a wide variety of standards, including Java 2 Enterprise Edition (J2EE), Java Server Pages (JSP), Servlets, Extensible Markup Language (XML), Secure Socket Layer (SSL), Enterprise Java Beans (EJB), Remote Method Invocation--Internet Inter-ORB Protocol (RMI-IIOP) servers, and/or Oracle DBMS. The systems and techniques described here leverage the robustness and acceptance of these technologies to deliver solutions that can scale across the entire enterprise.

Summary of Invention Paragraph (23):
[0021] In one implementation, an information server combines data from heterogeneous sources. The information server serves as middleware between applications and analysis modules, and the data sources. Each data source is associated with a data wrapper that publishes virtual tables of the information in the data source. An advantage of using a wrapper is that the data remains in the original location and the data source's native processing capabilities may be used to access the information. The wrapper may cache data that does not change very frequently to speed up subsequent queries.

Summary of Invention Paragraph (24):
[0022] The information server may include an accumulator that aggregates, normalizes and de-duplicates data from related data sources into a single universal data representation ("UDR") (see U.S. patent application Ser. No. 09/196,878, incorporated herein by reference) that can subsequently be queried and analyzed by applications. The accumulator de-duplicates data by removing duplicate or redundant data, normalizes data by applying algorithms to normalize the data against known reference values, and by applying domain-specific ontology to normalize the vocabulary across various data sources.

Summary of Invention Paragraph (27):
[0025] The processing server, which may be thought of as an analysis engine, may use a wrapper to wrap the "best" (e.g., the most appropriate for the context) of the available analysis tools into a single processing environment. These tools can be wrapped regardless of whether they are proprietary or in the public domain. The wrapper translates the data (e.g., now in UDR format) into any input format required by the various analysis tools. The tools may be located on the same machine as the processing server, in different hardware and software environments, or may be distributed over a network such as the Internet. The processing server's tool wrappers hide details, such as input and output formats, platform and location of each tool, and parameters required to run the tool, from the user and provide a consistent view of the tools to the user. Results of the analysis may be saved to the information server.

Summary of Invention Paragraph (28):
[0026] Applications may benefit from the processing server in many ways--the abstraction of the data access, the abstraction of the analysis execution, the transparency of the analysis location (local and remote tools), and/or the unified access of both data and results.

Summary of Invention Paragraph (30):
[0028] A visualization server, which is a specialized version of the processing server, provides a visualization framework by incorporating a variety of viewers, visualizers, and data mining tools. Each of these visualization tools has a wrapper that abstracts the tools to form a visualization framework that allows the user to view the outputs of queries or the results of analyses.

Summary of Invention Paragraph (31):
[0029] Various implementations may provide one or more of the following advantages. A query across multiple, heterogeneous data sources can be processed to produce transformed, normalized data that is optimized for each data source and that takes advantage of the data source's native processing capabilities to improve the results of the search. Both public and proprietary data stored in various locations and in different formats can be integrated, including relational databases, flat files, and Web (World Wide Web) and FTP (File Transfer Protocol) sites, in local and remote locations.

Summary of Invention Paragraph (32):
[0030] Heterogeneous data sources at different locations and in different formats can be searched and the results from the search can be integrated into a universal data representation. A query can be performed across several heterogeneous data sources with the query being optimized for each data source.

Summary of Invention Paragraph (38):
[0036] Additional data sources can be incorporated into an existing system with little or no changes to the system. A system can be expanded quickly by adding additional servers for increased capacity and additional nodes for multiple sites. A system can be configured so that public data is maintained externally and proprietary data is maintained behind a firewall.

Summary of Invention Paragraph (53):
[0051] A distributed data processing system may include an interface configured to receive a data processing request from a requesting entity, a processing server configured to provide access to one or more local data processing applications, one or more shadow processing servers, each shadow processing server configured to provide access to one or more remote data processing applications, and an application server, in communication with the processing server and the shadow processing server, and configured to fulfill the received data processing request by selectively accessing local and remote data processing applications in a manner that is transparent to the requesting entity. The interface configured to receive a data processing request from a

requesting entity may be a web server. Each shadow processing server may have a communications link for communicating with an interface at a remote data processing system. The shadow processing server may communicate with a servlet executing in a web server at the remote data processing system. Each shadow processing server may have an associated configuration file that identifies one or more remote data processing applications.

Summary of Invention Paragraph (54):
[0052] A distributed data acquisition system may include an interface configured to receive a data acquisition request from a requesting entity, an information server configured to provide access to one or more local data sources, one or more shadow information servers, each shadow information server configured to provide access to one or more remote data sources, and an application server, in communication with the information server and the shadow information server, and configured to fulfill the received data acquisition request by selectively accessing local and remote data sources in a manner that is transparent to the requesting entity.

Summary of Invention Paragraph (55):
[0053] A distributed data acquisition and processing system may include an interface configured to receive an information request from a requesting entity, a processing server configured to provide access to one or more local data processing applications, one or more shadow processing servers, each shadow processing server configured to provide access to one or more remote data processing applications, an information server configured to provide access to one or more local data sources, one or more shadow information servers, each shadow information server configured to provide access to one or more remote data sources, and an application server, in communication with the processing server, the shadow processing server, the information server, and the shadow information server, and configured to fulfill the received information request by selectively accessing local and remote data sources and local and remote data processing applications in a manner that is transparent to the requesting entity.

Brief Description of Drawings Paragraph (4):
[0058] FIGS. 3a and 3b are block diagrams of an information server.

Brief Description of Drawings Paragraph (7):
[0061] FIG. 5 is a block diagram of an application server, an information server, a processing server, and a visualization server.

Detail Description Paragraph (2):
[0066] FIG. 1 shows an implementation of an informatics platform. The platform combines heterogeneous data sources 22, analysis tools 18, and visualization applications 20 in a single framework. The platform may combine these heterogeneous entities without displacing existing systems that already use the sources, tools, or applications. The platform uses middleware engines, in this example, the information server 14, the processing server 16, and the visualization server 12. The information server 14 provides a semantically consistent view of the data from several dynamic, heterogeneous data sources 22. This information is provided in the form of a virtual database 10, which can be accessed by the processing server 16 and the visualization server 12 through the information server 14. (Although FIG. 1 shows the virtual database 10 as a separate entity from the information server 14, in a typical implementation, virtual database 10 may reside within the information server 14.) The processing server 16 is able to combine various different types of analysis tools 18, including public domain tools, third party solutions, and proprietary custom-developed tools, in a single processing environment thereby providing "virtual compute services" that represent the best-of-class analysis tools. The visualization server 12 can combine a variety of viewers, visualizers, and data mining tools 20 into a visualization framework. The viewing tools 20 are abstracted by the visualization server 12 to provide datatype-specific visualization services that can be invoked by an application to view the results of queries or analyses. The platform may be made platform independent, for example, by implementing it in Java or an equivalent language.

Detail Description Paragraph (3):
[0067] As shown in FIG. 2, a basic system architecture (which will be explained in further detail in reference to FIGS. 5 and 6) may include a web server 34, which gives users an interface to manage data, execute tasks, and view results. The web server 34 separates the user interface from the application logic contained in an application server 36 (explained in greater detail in reference to FIG. 6). The application server 36 hosts application logic and provides a link between the web server 34 and the visualization server 12, the processing server 16, and the information server 14. The

information server 14 hosts and manages access to the virtual database 10.

Detail Description Paragraph (4):
[0068] FIG. 3a is a simplified view of an information server 14. The information server 14 may include one or more data wrappers 24 which are discussed in more detail below under the heading: Anatomy of a Data Wrapper. As illustrated, wrappers 24a, 24b, 24c, and 24d each corresponds to an associated data source 22 (namely, sources 22a, 22b, 22c, 22d) that is accessed through the information server 14. Data sources 22 may be in the form of flat text files, Excel spreadsheets, extensible Markup Language XML (Extensible Markup Language) formatted documents, relational databases, data feeds from proprietary servers, and web-based data sources. For instance, database 22a has a corresponding data wrapper 24a. Similarly, flat file 22b, XML document 22c, and Web site 22d each has a corresponding wrapper (24b, 24c, and 24d, respectively). (This illustration shows four data sources 22; however, an information server can accommodate any number of heterogeneous data sources, each having a corresponding wrapper.)

Detail Description Paragraph (5):
[0069] Data wrappers 24 access data from the associated data source's original location and in the original format, and isolate applications receiving the data from the protocols and formats required to interact with the data sources 22. Data wrappers are generally constructed to take advantage of any native query and processing capabilities of their respective data sources in accessing information. A data wrapper 24, optionally, may cache information to a local wrapper cache 38 to improve data access speed on subsequent queries. Typically, each data wrapper 24 would have its own associated cache 38. A wrapper cache 38 can be enabled or disabled depending on each data source; generally, only data that does not change very frequently should be cached. Caching typically is most beneficial when access to the data source is slow--for example, caching data from a relational database that has a very fast access time may be less beneficial than caching data from an instrument that has slow data access. A wrapper cache 38 can be implemented in a relational database local to the information server 14, for example, within the same local area network as the information server. Each record stored in the cache is assigned a Time-to-Live (TTL) value that specifies how long (in seconds) that record should remain in the cache before it expires. Expired records are automatically removed from the cache.

Detail Description Paragraph (7):
[0071] Data wrappers 24 may be implemented with an error detection and notification mechanism. This mechanism in a wrapper detects changes in the location or structure of the data for a corresponding data source. When a change is detected that cannot be handled by the wrapper, the wrapper stops providing data and it transmits a notification (i.e., a request for repair) to a self-healing manager (SHM) component. The SHM contacts a support site) and looks for updates to the wrapper. The notification can be transmitted using any messaging protocol such as Simple Mail Transfer Protocol (SMTP), or HyperText Transport Protocol (HTTP) post.

Detail Description Paragraph (8):
[0072] The self-healing manager (SHM) may be implemented as a separate process running on a computer in communication, either locally or remotely, with the platform. The SHM continually polls until an update is available. The frequency of the polling is a tunable parameter and depends on the context of the application. When the SHM receives a request for repair, it first determines whether an update exists for the wrapper in question. If there is, the update is downloaded and installed by the SHM. Wrapper updates can be downloaded from the information server and installed to replace the defective wrapper even while the wrapper is running. If no update is available, the SHM notifies a support site, so that support personnel will prepare an update. When the update is ready, it is posted by the support personnel to the support site so that it can be downloaded and installed by the SHM on the next polling cycle, as has been described above. When the wrapper is updated, the wrapper resumes providing data. For each subsequent error that is detected, the wrapper sends another notification and takes itself off-line until it is has been replaced by a replacement wrapper capable of processing the data without error. The self-healing mechanism is not limited to wrappers in the information server 14--it is also available for wrappers on the processing server 16 and visualization server 12, and accumulators as discussed below.

Detail Description Paragraph (9):
[0073] An accumulator 28 aggregates virtual tables 26 into a single universal data representation (UDR) 32. Further details of accumulators are discussed below under the heading: Anatomy of an Accumulator. An information server may have more than one accumulator. For example, different accumulators may be required for different types of

data being provided by an information server; or, one accumulator may be configured to receive as an input a UDR provided by another accumulator. In general, an information server may include as many accumulators as appropriate to fulfill its data-providing function. Moreover, these accumulators may be arranged in multiple, interconnected levels to aggregate and normalize the gathered data as desired. An accumulator optionally may have a local cache 30 to store frequently requested and relatively static data.

Detail Description Paragraph (11):
[0075] An accumulator not only aggregates the data, but it also may normalize and de-duplicate the aggregated data. Normalization may take place at two levels. The first, data normalization, applies algorithms to normalize the data against known reference values. The type and nature of algorithms to be used for data normalization is highly context specific and depends on the nature of the data to be normalized. Vocabulary normalization, the second form of normalization performed by the accumulator, applies a domain-specific ontology to normalize the vocabulary across data sources. For example, if one data source refers to "human" data while source refers to "Homo sapiens" data, the accumulator will employ a synonym-based replacement of some data to normalize the sources (i.e., replace "Homo sapiens" with "human"). In another example, if one data source has a column labeled "Sequence ID" and another data source has a column labeled "Accession Number," the accumulator logic recognizes these are identical concepts and will take the different column names and map them to a single column with a single name.

Detail Description Paragraph (13):
[0077] FIG. 3b offers a more detailed view of an information server 14. The information server 14 contains four main modules--a data engine 70, a data formatter 72, a query engine 74, and a remote data connector 76.

Detail Description Paragraph (18):
[0082] FIG. 3c shows a block diagram for a process of performing a query. A user query 300 is received by an information server 14. The query engine at the information server 14 evaluates the query 300 and directs it to the UDR 302 output of the accumulator 304. The query executor of accumulator 304 receives the query, evaluates the query to determine what information it needs from each of the virtual tables that are inputs to the accumulator, and creates new queries 306, 308, 310 that will be sent to associated virtual tables 316, 318, 320. Each of the wrappers 326, 328, 330 receives its respective query 300, 306, 308, 310, and evaluates the query to determine what information needs to be retrieved from the wrapped data sources 311, 313, 315 Each wrapper then creates queries 336, 338, 340 in the native query language of each data source 311, 313, 315 and sends it to that data source. The output of the queries 336, 338, 340 produce a list of records 346, 348, 350. The results are then transformed by the wrapper into a physical recordset 356, 358, 360 in the virtual table output format 316, 318, 320. If a detail record exists in the wrapper cache 327,329,331 the record is retrieved out of the cache and stored in the corresponding recordset 356, 358, 360. Otherwise, the detail record is retrieved directly from the data source 311, 313, 315 and transformed to the corresponding recordset 356, 358, 360.

Detail Description Paragraph (20):
[0084] As shown in FIG. 4, a search begins when a user submits a query through a user interface to the web server (step 120). The web server passes this query to the application server (step 122), a process described in greater detail below in reference to FIG. 6. The application server then passes the query to the local information server in SQL format (step 124), a process also described in reference to FIG. 6. The query is then passed to the local information server's query engine for evaluation (step 126). The query engine translates the query into calls to individual accumulators and/or wrappers contained in the data engine (step 128).

Detail Description Paragraph (24):
[0088] The results of this query are aggregated by the accumulator (step 136). The information server's data engine retrieves the results from the accumulator (step 138). The information server's data formatter formats the results into any required format and stores them for subsequent analysis (step 140).

Detail Description Paragraph (25):
[0089] If a query is requesting data that is coming from remote information servers, the Remote data connector 76 is used to pass the data request to a registered shadow information server to retrieve results from the remote information server (this process will be discussed in detail in reference to FIG. 8), and manager the satisfactory

completion of the request. A data request is any request to retrieve data from the information server. It could be a query, or merely a request to retrieve all the results of an analysis by name. The data requester, e.g., an application, therefore only has to deal with the local information server but can transparently obtain data from any remote server.

Detail Description Paragraph (26):
[0090] As illustrated in FIG. 5, the data obtained by the information server 14 and made available in the UDR 32 can be analyzed by the processing server 16 or viewed by the visualization server 12. Virtually any number of analysis tools 18 (illustrated as tools 18a, 18b, 18c) can be linked by the processing server 16. The analysis tools 18 (e.g., data processing applications) may require data in different formats and may run on different platforms, such as Solaris on Sun Enterprise, WinNT/2000 and Linux on Intel, Tru64 on Compaq AlphaServer, and IRIX on SGI Origin or proprietary hardware platforms such as the Paracel GeneMatcher or TimeLogic DeCypher. Analysis tools do not have to reside locally in order to be incorporated into the processing server--Web-accessible tools can also be transparently incorporated into the processing server to form a compute service.

Detail Description Paragraph (27):
[0091] The processing server 16 requests data in the UDR 32 through the information server connector 19, an API for communicating with the information server. Application wrappers 40 specifically written for each tool 18 (so, in the illustration, tool 18a has a corresponding wrapper 40a, tool 18b corresponds with wrapper 40b, tool 18c corresponds with wrapper 40c) convert data into desired input format of the corresponding tool 18 by data transformation rules when necessary. The particular data transformation rules are application-specific rules necessary to prepare the inputs for the tool to run correctly. The processing server 16, using the wrappers 40 provides a consistent interface for the analysis tools and hides from the invoking application the execution details of the analysis tools 18, such as input formats, output formats, platform, and parameters required to run the tool 18. The interface provided by the processing server is application-specific and can be any implementation that effectively communicates the parameters and output format between the application and the tools; in one embodiment, the interface encodes the parameters in XML. As will be shown below in FIG. 9, tools 18 do not need to be local but may be transparently incorporated into the processing server 16 from remote locations.

Detail Description Paragraph (28):
[0092] Results of each analysis are stored in the tool's native format but wrapped as an object, which may later be converted into the UDR by the information server 14 so that other analysis tools 18 may access the results as part of an analysis workflow. An analysis workflow is a pipelined way to chain together a group of tasks wherein the output of one task can be used as the input into another task to increase throughput of the analysis.

Detail Description Paragraph (29):
[0093] The application server 36 keeps a log of a user's actions in an audit trail 100, which may be as simple as a text file or something more structured, such as a relational database. This database can be used to generate an analysis workflow.

Detail Description Paragraph (30):
[0094] The visualization server 12 is a special implementation of the processing server 16. Viewers, visualizers, and data mining tools 20 (for example, desktop tools, Java applets, and viewers of data formatted in a markup language such as HyperText Markup Language (HTML), Postscript, PDF or any other desired format) are incorporated into a visualization framework to form datatype-specific visualization services that can be invoked by an application as a result of a user request to view the output of a query. The visualization framework provides an endpoint or destination for the query output. Wrappers 46 specific to each different visualization tool 20 abstract the tools 20 to form the visualization framework, illustrated as wrapper 46a for tool 20a, wrapper 46b for tool 20b, and wrapper 46c for tool 20c.

Detail Description Paragraph (31):
[0095] FIG. 6 illustrates a specific implementation for task execution of the basic architecture described above in reference to FIG. 2. Web server 34 provides an interface that users can use to manage data, execute tasks, and view results. The web server 34 separates the user interface from the application logic contained in the application server 36. The web interface is implemented using Java Server Pages (JSPs) 48, which enable generation of dynamic web pages and which make calls to the

application server 36 for executing the application logic. In this implementation, the application logic is realized in an Enterprise JavaBeans (EJB) container 56. The web server contains an HTML module 54, which contains static Web page templates to be combined with dynamic content. A Java servlet 50 receives requests from clients, i.e., system users. An EJB stub 52 then relays the request to the application server 36.

Detail Description Paragraph (32):
[0096] The application server 36, as noted above, hosts the application logic and provides a link between the web server 34 and the information, processing, and visualization servers 14, 16, 12. The application logic components in this embodiment are deployed as Enterprise JavaBeans in the EJB container 56. Available processing or visualization servers 16, 12 are listed in a server registry bean 60 on the application server. Upon startup of a processing server, the processing server is registered with a Java Naming and Directory Interface (JNDI) service 68 on the application server. During the registration process, the processing server tells the application server which tools are available on the processing server.

Detail Description Paragraph (33):
[0097] When a request to execute a task comes from the web server 34 through the EJB stub 52, the web server 34 uses the EJB's remote interface to connect to a task manager bean 58 on the application server. The task manager bean 58 instantiates and passes on all appropriate initialization parameters to a task bean 64. When initialization is complete and the task is ready to run, the task manager bean 58 is notified to add the task to a queue of tasks on the application server. The task manager bean 58 then checks a work queue for each processing server 16 that is capable of performing the task and uses a load-balancing approach to determine which processing server is available to perform the task. If no processing server 16 is available, the task remains in the task queue until assigned to a processing server 16. The task manager bean 58 notifies the requestor that the task has been queued for execution. However, if a processing server 16 is available, the task manager bean 58 sends a message to one of the processing servers 16 to execute the task. The message is received by a message listener thread 134 in the processing server 16 and threads 42 are created for the task in the task execution engine 51. The status of the task is tracked by the task monitor thread 63 within the processing server 16. The requestor can request to receive periodic notices regarding the task status.

Detail Description Paragraph (34):
[0098] A workflow bean 62 in the application server 36 tracks statistics, such as the amount of time in a job queue, time-to-completion, and error states for all running tasks.

Detail Description Paragraph (35):
[0099] The elements that have been described also can be implemented to run tasks on the information and visualization servers 14, 12.

Detail Description Paragraph (36):
[0100] FIG. 7 illustrates the system architecture at a local node 98. The architecture is extended to include shadow servers 80, 88 serving as proxies for events happening on a remote node 100. The shadow processing server 80 and the shadow information server 88 are responsible for accessing tools and data, respectively, located on one or more remote nodes 100; optimally, each shadow server is responsible for only a single remote node 100. Multiple shadow servers may exist in one node.

Detail Description Paragraph (37):
[0101] The shadow servers 80, 88 each have a configuration file 78, 97 containing authentication credentials for communicating with the servers on remote node 100. The configuration file 78, 97 also specifies the tools/data resident on the remote node 100 and this information is provided to the application server 36 during registration of the shadow processing server 80 with the application server 36. The registration process is the same as with the local processing server discussed above.

Detail Description Paragraph (38):
[0102] The following describes how a shadow processing server 80 can be used to access a tool (e.g., a data processing or analysis application) located on a remote node 100 access: When the application server 36 at the local node 98 receives from web server 34 a user request to access a Tool 4, a task manager EJB on the application server 36 consults a registry of processing servers (maintained by application server 36 and containing both local and shadow servers) to determine which processing server can provide Tool 4. In the case where Tool 4 resides on a remote node 100, the task manager

EJB assigns the task to the shadow processing server 80 responsible for remote node 100.

Detail Description Paragraph (39):
[0103] Upon receiving the request, the shadow processing server 80 constructs an XML (Extensible Markup Language) message describing the task and uses HTTPS (HyperText Transmission Protocol, Secure) to forward the XML message to a servlet 86 on the web server of the remote node 100. The servlet 86, upon receiving the XML message from the shadow processing server 80, reads the XML message, decomposes the message into a local task, and responds back to the shadow processing server 80 with another XML message containing the data requirements for performing the task.

Detail Description Paragraph (40):
[0104] The shadow processing server 80 receives the responding message from servlet 86, decodes the message, and communicates with local information server 14 to obtain the input data and send it using an HTTPS POST operation to a data handling servlet 94 of the remote node 100. The data handling servlet 94 reads the data streams and caches the data at the remote information server 92 on the remote node 100, thereby satisfying the input requirements for the task. The data handling servlet 94 returns a status to the shadow processing server 80, which then sends another XML message to the remote application servlet 86 to schedule the task for execution on the remote node 100.

Detail Description Paragraph (41):
[0105] The servlet 86 connects to the remote application server 102 and communicates with task manager at node 100 to create a task and schedule it to run on the remote processing server 104. The shadow processing server 80 (which is responsible for reporting the task status back to application server 36) continually polls servlet 86 for the status of the task. This polling occurs in the form of an XML message. Upon receiving the status request, the servlet 86 asks the application server 102 for status and responds back to the shadow processing server 80. The shadow processing server 80 uses the status received from the servlet 86 to update the task status for the task assigned to it from application server 36. When the shadow processing server 80 receives notice that the task is complete, the shadow processing server 80 requests the resulting data from the data handling servlet 94. The servlet 94 communicates with the remote information server 92 to retrieve the results and to pass them to the shadow processing server 80. The shadow processing server 80 may request the local information server 14 to store the results and then informs the application server 36 that the task is complete.

Detail Description Paragraph (42):
[0106] The following describes how the shadow information server 88 can be used to access data residing on a remote node 100. All user requests to access data are sent first to the local information server 14. Then, if some or all of the requested data is non-local, the local information server 14 passes the request to one or more shadow information servers 88 (depending on where the non-local data is), each of which interacts with a remote information server 92 to obtain the requested remote data from one or more remote data sources 90 connected to the remote information server 92. A remote information server 92 contains the same modules as the information server 14, described above, and processes queries in the same manner.

Detail Description Paragraph (43):
[0107] The local information server 14 has a remote data connector 76, which the server uses to communicate with one or more shadow information servers 88. The shadow information server 88 formats data requests as XML messages and passes the message via HTTPS to a data handling servlet 94 on the remote node 100. The data handling servlet 94 receives the XML messages, decodes the message, and sends the request to the remote information server 92. Servlet 94 authenticates the messages received from shadow information server 88, communicates with the remote information server 92, and handles the data transmission between the shadow server 88 and the remote information server 92. The remote information server 92, when it receives a data request from the data handling servlet 94, completes the data request, and sends the results back to the data handling servlet 94. The data handling servlet 94 returns the data to the shadow information server 88 as a response to the XML message that the servlet 94 received. The shadow server 88 caches the data locally and sends the data through the remote data connector to the information server 14.

Detail Description Paragraph (44):
[0108] FIG. 8 is a block diagram showing an example of a split node distributed over three sites 900, 902 and 904. As used herein, a split node is one in which the

available analysis functionality and/or available data sources are distributed across two or more sites. Such a configuration may be used, for example, in a distributed enterprise having facilities in three different geographic locations such as London, New York and Los Angeles. Although each site has only a subset of the enterprise's available tools and/or data sources locally present, a user at any of the sites has virtual and transparent access to all of the enterprise's tools and data sources through a system of shadow servers. In FIG. 8, tools and data sources that are locally present are shown in solid lines while tools and data sources that are virtually present (i.e., located remotely but made transparently available) are shown in dotted lines.

Detail Description Paragraph (45):
[0109] As shown in FIG. 8, for example, the enterprise's New York site 900 has only tools D, B, E and data sources X, Y, Z physically present at site 900. A user at the New York site 900 may access the tools D, B, E and/or the data sources X, Y, Z by interfacing directly with a web server 916, which receives the user's data or processing request and passes it to the application server 911. The application server 911 in turn fulfills the request by initiating a task to selectively access the processing server 915 and/or the information server 913 as appropriate.

Detail Description Paragraph (46):
[0110] In addition, shadow servers 903, 905, 907, 909 at the New York site 900 enable a user at that site to transparently and seamlessly access any of the tools A, B, C or data sources T, U, V at the Los Angeles site 902 and/or any of the tools A, F, G or data sources Q, R, S at the London site 904. More particularly, the New York site 900 includes a separate shadow processing server 903, 905 for each of the other sites 902 and 904, respectively. In the manner described with reference to FIG. 7, the LA shadow processing server 903 registers with the application server 911 to inform the application server 911 that tools A, B, C are available at the Los Angeles site 902. Consequently, the tools present at the Los Angeles site 902 are presented to a user at the New York site 900 as being available for usage. Because the availability of these remote tools is presented to the user in the same manner as the availability of the local tools (that is, the remote tools are presented in a location-transparent manner), the user at the New York site 900 may be unaware that the tools are located remotely.

Detail Description Paragraph (47):
[0111] Connections between servers across site boundaries are not shown in FIG. 8 for the sake of clarity. However, each shadow server at a site has a communications connection to a servlet executing in a web server at a corresponding remote site. For example, the shadow processing server (LA) 903 at site 900 has a connection to a servlet 927 at site 902 and the shadow information server (LA) 907 at site 900 has a connection to a servlet 929 at site 902. Similarly, the shadow processing server (NY) 921 at site 902 has a connection to a servlet 931 at site 900 and the shadow information server (NY) 923 has a connection to a servlet 933 at site 900. Analogous connections exist for sites 902-904 and for sites 900-904 between shadow servers and associated servlets. A request from a remote site received by a servlet in a web browser, whether for data or processing, is passed on to that site's application server, which in turn initiates a task to fulfill the request. Request results and/or status subsequently are returned to the servlet, which communicates the results/status back to the originating shadow server. In this process, the application server effectively is unaware that the request was originated remotely and thus acts to fulfill the request in the same manner as if it were initiated locally. In this way, each site in the split node can make all of the enterprise's tools and data sources available, either physically or virtually, to users at any of the sites.

Detail Description Paragraph (52):
[0116] UDR U4, in addition to being fed into Accumulator5 954, could also be used as input into one or more tools or applications. An application wrapper AppA 958 takes input data from UDR U4 and converts the data into T6, which represents the input format required by a particular application or tool. Once the tool has completed its execution, the output T3A can be stored in one or more formats for use by one or more visualization servers. Alternatively, output T3A can be re-used as input back into the information servers, here shown by feedback loop 962. To execute the feedback loop 962, AppA 958 converts T3A into T3, which is the input format for Wrappers3 964. AppA 960 then stores T3 in the location where Wrapper3 typically retrieves data. In a second iteration, Wrapper3 could retrieve the new T3 and pass it to Accumulator4 956 to form a new UDR U4.

Detail Description Paragraph (54):

[0118] The components and techniques described here may be implemented in digital electronic circuitry, or in computer hardware, firmware, software, or in combinations of them. An apparatus can be implemented in a computer program product tangibly embodied in a machine-readable storage device for execution by a programmable processor; and method steps can be performed by a programmable processor executing a program of instructions to perform functions by operating on input data and generating output. These techniques may be implemented advantageously in one or more computer programs that are executable on a programmable system including at least one programmable processor coupled to receive data and instructions from, and to transmit data and instructions to, a data storage system, at least one input device, and at least one output device. Each computer program may be implemented in a high-level procedural or object-oriented programming language, or in assembly or machine language if desired; and in any case, the language can be a compiled or interpreted language. Suitable processors include, by way of example, both general and special purpose microprocessors. Generally, a processor will receive instructions and data from a read-only memory and/or a random access memory. The essential elements of a computer are a processor for executing instructions and a memory. Generally, a computer will include one or more mass storage devices for storing data files; such devices include magnetic disks, such as internal hard disks and removable disks; magneto-optical disks; and optical disks. Storage devices suitable for tangibly embodying computer program instructions and data include all forms of non-volatile memory, including by way of example semiconductor memory devices, such as EPROM, EEPROM, and flash memory devices; magnetic disks such as internal hard disks and removable disks; magneto-optical disks; and CD-ROM disks. Any of the foregoing can be supplemented by, or incorporated in, ASICs (application-specific integrated circuits).

Detail Description Paragraph (68):
[0132] 5. Error handling--In order to maintain the uptime of a system, each component must be able to sense errors or changes to the data source. Errors can be of four forms: system errors, hard errors, soft errors, or warnings. (1) The system errors are such things as HTTP 500 Server Error, Connection Timeout, DNS (Domain Name Service) Entry not found, etc. Errors that have nothing to do with the data that is trying to be accessed, rather, the system that the wrapper is trying to access has some error condition that prevents the successful extraction of the data. (2) The hard errors are such things as table name not found, field not found, URL gives HTTP 404 Not Found error, etc. These errors would cause the wrapper to "break", and in need of repair through the self-healing manager. (3) Soft errors are such things as new fields that are discovered in the tables of a database wrapper (through a database reverse engineering process), or on a file or web page where new fields appear in the data buffer as part of parsing a structured document. These errors, although not critical to the operation of the wrapper, may need human review to check for the semantic meaning of the new fields and their importance for inclusion into the UDR. (4) Warnings are solely for notification purposes; the system does not perform any action in response to the warning.

Detail Description Paragraph (69):
[0133] a. Self-healing manager registration--Each component is registered with a self-healing manager that is responsible for maintaining the correct state of the components. The information that is registered with the self-healing manager is the component class path (i.e. com.adaapt.wrapper.web.NCBIEntrezWebWrapper), the version of the component in Major.Minor.Revision format (i.e. 1.0.4), the author's name and email address, and the support server that is responsible for keeping this component up to date (i.e. support.entigen.com/patch/patchserver).

Detail Description Paragraph (130):
[0194] 8. Output Data Model--Results of each analysis are stored in the tool's native format but wrapped to produce a virtual table which is later converted into the UDR by the information server 14.

CLAIMS:

32. A distributed data processing system comprising: an interface configured to receive a data processing request from a requesting entity; a processing server configured to provide access to one or more local data processing applications; one or more shadow processing servers, each shadow processing server configured to provide access to one or more remote data processing applications; and an application server, in communication with the processing server and the shadow processing server, and configured to fulfill the received data processing request by selectively accessing local and remote data processing applications in a manner that is transparent to the

requesting entity.

33. The system of claim 32 wherein the interface configured to receive a data processing request from a requesting entity comprises a web server.

34. The system of claim 32 wherein each shadow processing server has a communications link for communicating with an interface at a remote data processing system.

35. The system of claim 34 wherein the shadow processing server communicates with a servlet executing in a web server at the remote data processing system.

36. The system of claim 32 wherein each shadow processing server has an associated configuration file that identifies one or more remote data processing applications.

37. A distributed data acquisition system comprising: an interface configured to receive a data acquisition request from a requesting entity; an information server configured to provide access to one or more local data sources; one or more shadow information servers, each shadow information server configured to provide access to one or more remote data sources; and an application server, in communication with the information server and the shadow information server, and configured to fulfill the received data acquisition request by selectively accessing local and remote data sources in a manner that is transparent to the requesting entity.

38. The system of claim 37 wherein the interface configured to receive a data acquisition request from a requesting entity comprises a web server.

39. The system of claim 37 wherein each shadow information server has a communications link for communicating with an interface at a remote data processing system.

40. The system of claim 39 wherein the shadow information server communicates with a servlet executing in a web server at the remote data acquisition system.

41. The system of claim 37 wherein each shadow information server has an associated configuration file that identifies one or more remote data source.

42. A distributed data acquisition and processing system comprising: an interface configured to receive an information request from a requesting entity; a processing server configured to provide access to one or more local data processing applications; one or more shadow processing servers, each shadow processing server configured to provide access to one or more remote data processing applications; an information server configured to provide access to one or more local data sources; one or more shadow information servers, each shadow information server configured to provide access to one or more remote data sources; and an application server, in communication with the processing server, the shadow processing server, the information server, and the shadow information server, and configured to fulfill the received information request by selectively accessing local and remote data sources and local and remote data processing applications in a manner that is transparent to the requesting entity.

WEST

Help | Logout | Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Term	Documents
ONTOLOGY	376
ONTOLOGIES	149
ONTOLOGYS	2
(ONTOLOGY AND 15).USPT,PGPB,JPAB,EPAB,DWPI,TDBD.	2
(L15 AND ONTOLOGY).USPT,PGPB,JPAB,EPAB,DWPI,TDBD.	2

Database: US Patents Full-Text Database
US Pre-Grant Publication Full-Text Database
JPO Abstracts Database
EPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L16

Refine Search

Recall Text | Clear

Search History

DATE: Friday, August 22, 2003 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L16	L15 and ontology	2	L16
L15	L14 and (search$ near (result or response))	283	L15
L14	L13 and transmit$	494	L14
L13	L12 and (search$ near request$)	601	L13
L12	(receive near request) and client and server	8492	L12
L11	L9 and (transmit$ near search$ near request$)	5	L11
L10	L9 and (transmit$ near search$ near request4)	0	L10
L9	(receive near request) same client	4331	L9
L8	L7 and (ontology near description)	0	L8
L7	ontology near server	17	L7
L6	l2 and ontology	6	L6
L5	L4 and ontology	0	L5
L4	L3 and (url near result$)	56	L4
L3	L2 and (request$ or query or search$)	1686	L3
L2	L1 and (web near server) and (web near client)	1734	L2
L1	client near server	30905	L1

END OF SEARCH HISTORY